Reasoning about actions with Temporal Answer Sets

Reasoning about Actions with Temporal Answer Sets

L. Giordano, A. Martelli and D. Theseider Dupre'
(preliminary version)

Published in:

Theory and Practice of Logic Programming 13(2) 201-225 (2013)

Reasoning about Actions with Temporal Answer Sets

Laura Giordano
Dipartimento di Informatica, Università del Piemonte Orientale, Italy
laura@mfn.unipmn.it

Alberto Martelli
Dipartimento di Informatica, Università di Torino, Italy
mrt@di.unito.it

Daniele Theseider Dupré
Dipartimento di Informatica, Università del Piemonte Orientale, Italy
dtd@mfn.unipmn.it

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Abstract

In this paper, we combine Answer Set Programming (ASP) with Dynamic Linear Time Temporal Logic (DLTL) to define a temporal logic programming language for reasoning about complex actions and infinite computations. DLTL extends propositional temporal logic of linear time with regular programs of propositional dynamic logic, which are used for indexing temporal modalities. The action language allows general DLTL formulas to be included in domain descriptions to constrain the space of possible extensions. We introduce a notion of Temporal Answer Set for domain descriptions, based on the usual notion of Answer Set. Also, we provide a translation of domain descriptions into standard ASP and we use Bounded Model Checking techniques for the verification of DLTL constraints.

1 Introduction

Temporal logic is one of the main tools used in the verification of dynamic systems. In the last decades, temporal logic has been widely used also in AI in the context of planning, diagnosis, web service verification, agent interaction and, in general, in most of those areas having to do with some form of reasoning about actions.

The need of temporally extended goals in the context of planning has been first motivated by Bacchus and Kabanza (Bacchus and Kabanza 1998), Kabanza et al. (Kabanza et al. 1997) and by Giunchiglia and Traverso (Giunchiglia and Traverso 1999). In particular, (Giunchiglia and Traverso 1999) developed the idea of planning as model checking in a temporal logic, where the properties of planning domains are formalized as temporal formulas in CTL. In general, temporal formulas can be usefully exploited both in the specification of a domain and in the verification of its properties. This has been done, for instance, for modeling the interaction of services on the web (Pistore et al. 2005), as well

as for the specification and verification of agent communication protocols (Giordano et al. 2007). Recently, Claßen and Lakemeyer (Claßen and Lakemeyer 2008) have introduced a second order extension of the temporal logic CTL*, $\mathcal{ESG}$, to express and reason about non-terminating Golog programs. The ability to capture infinite computations is important as agents and robots usually fulfill non-terminating tasks.

In this paper we combine Answer Set Programming (ASP) (Gelfond 2007) with Dynamic Linear Time Temporal Logic (DLTL) (Henriksen and Thiagarajan 1999) to define a temporal logic programming language for reasoning about complex actions and infinite computations. DLTL extends propositional temporal logic of linear time with regular programs of propositional dynamic logic, which are used for indexing temporal modalities. Allowing program expressions within temporal formulas and including arbitrary temporal formulas in domain descriptions provides a simple way of constraining the (possibly infinite) evolutions of a system, as in PDL.

To combine ASP and DLTL, we define a temporal extension of ASP by allowing temporal modalities to occur within rules and we introduce a notion of Temporal Answer Set, which captures the temporal dimension of the language as a linear structure and naturally allows to deal with infinite computations.

A domain description is defined to consist of two parts: a set of temporal rules (action laws, causal laws, etc.) and of a set of constraint (arbitrary DLTL formulas). The temporal answer sets of the rules in the domain description which also satisfy the constraints are defined to be the extensions of the domain description.

We provide a translation into standard ASP for the action laws, causal laws, etc. of the domain description. The temporal answer sets of an action theory can then be computed as the standard answer sets of the translation.

To compute the extensions of a domain description, the temporal constraints, which are part of the domain description, are evaluated over temporal answer sets using *bounded model checking* techniques (Biere et al. 2003). The approach proposed for the verification of DLTL formulas extends the one developed in (Heljanko and Niemelä 2003) for bounded LTL model checking with Stable Models.

The outline of the paper is as follows. In Section 2, we recall the temporal logic DLTL. In Section 3, we introduce our action theory in temporal ASP. In Section 4, we define the notions of temporal answer set and extension of a domain description. Section 5 describes the reasoning tasks, while Sections 6 and 7 describe the model checking problem and provide a translation of temporal domain descriptions into ASP. Section 8 provides the conclusions and the related work.

2 Dynamic Linear Time Temporal Logic

In this section we briefly define the syntax and semantics of DLTL as introduced in (Henriksen and Thiagarajan 1999). In such a linear time temporal logic the next state modality is indexed by actions. Moreover, (and this is the extension to LTL) the until operator $\mathcal{U}^\pi$ is indexed by a program π as in Propositional Dynamic Logic (PDL). In addition to the usual $\Box$ (always) and $\Diamond$ (eventually) temporal modalities of LTL, new modalities $[\pi]$ and $\langle\pi\rangle$ are allowed. Informally, a formula $[\pi]\alpha$ is true in a world w of a linear temporal model if α holds in all the worlds of the model which are reachable from w through any execution of

the program π. A formula $\langle\pi\rangle\alpha$ is true in a world w of a linear temporal model if there exists a world of the model reachable from w through an execution of the program π, in which α holds. The program π can be any regular expression built from atomic actions using sequence (;), non-deterministic choice ($+$) and finite iteration ($*$). The usual modalities $\Box$, $\Diamond$ and $\bigcirc$ (next) of LTL are definable.

Let Σ be a finite non-empty alphabet. The members of Σ are actions. Let Σ^* and Σ^ω be the set of finite and infinite words on Σ, where $\omega = \{0, 1, 2, \ldots\}$. Let $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. We denote by σ, σ' the words over Σ^ω and by τ, τ' the words over Σ^*. We denote by *prf(u)* the set of finite prefixes of u. Moreover, we denote by $\leq$ the usual prefix ordering over Σ^* and, for $u \in \Sigma^\infty$, namely, we define $\tau \leq \tau'$ iff $\exists\tau''$ such that $\tau\tau'' = \tau'$ and we define $\tau < \tau'$ iff $\tau \leq \tau'$ and $\tau \neq \tau'$.

The set of programs (regular expressions) $Prg(\Sigma)$ generated by Σ is:

$$Prg(\Sigma) ::= a \mid \pi_1 + \pi_2 \mid \pi_1; \pi_2 \mid \pi^*,$$

where $a \in \Sigma$ and π_1, π_2, π range over $Prg(\Sigma)$. A set of finite words is associated with each program by the mapping $[[]] : Prg(\Sigma) \to 2^{\Sigma^*}$, which is defined as follows:

- $[[a]] = \{a\}$;
- $[[\pi_1 + \pi_2]] = [[\pi_1]] \cup [[\pi_2]]$;
- $[[\pi_1; \pi_2]] = \{\tau_1\tau_2 \mid \tau_1 \in [[\pi_1]] \text{ and } \tau_2 \in [[\pi_2]]\}$;
- $[[\pi^*]] = \bigcup[[\pi^i]]$, where
  - $[[\pi^0]] = \{\varepsilon\}$
  - $[[\pi^{i+1}]] = \{\tau_1\tau_2 \mid \tau_1 \in [[\pi]] \text{ and } \tau_2 \in [[\pi^i]]\}$, for every $i \in \omega$.

where ε is the empty word (the empty action sequence).

Let $\mathcal{P} = \{p_1, p_2, \ldots\}$ be a countable set of atomic propositions containing $\top$ and $\bot$ and let DLTL(Σ) ::= $p \mid \neg\alpha \mid \alpha \vee \beta \mid \alpha\mathcal{U}^\pi\beta$, where $p \in \mathcal{P}$ and α, β range over DLTL(Σ).

A model of DLTL(Σ) is a pair $M = (\sigma, V)$ where $\sigma \in \Sigma^\omega$ and $V : \mathit{prf}(\sigma) \to 2^{\mathcal{P}}$ is a valuation function. Given a model $M = (\sigma, V)$, a finite word $\tau \in \mathit{prf}(\sigma)$ and a formula α, the satisfiability of a formula α at τ in M, written $M, \tau \models \alpha$, is defined as follows:

- $M, \tau \models \top$;
- $M, \tau \not\models \bot$;
- $M, \tau \models p$ iff $p \in V(\tau)$;
- $M, \tau \models \neg\alpha$ iff $M, \tau \not\models \alpha$;
- $M, \tau \models \alpha \vee \beta$ iff $M, \tau \models \alpha$ or $M, \tau \models \beta$;
- $M, \tau \models \alpha\mathcal{U}^\pi\beta$ iff there exists $\tau' \in [[\pi]]$ such that $\tau\tau' \in \mathit{prf}(\sigma)$ and $M, \tau\tau' \models \beta$. Moreover, for every τ'' such that $\varepsilon \leq \tau'' < \tau'$, $M, \tau\tau'' \models \alpha$.

A formula α is satisfiable iff there is a model $M = (\sigma, V)$ and a finite word $\tau \in \mathit{prf}(\sigma)$ such that $M, \tau \models \alpha$. The formula $\alpha\mathcal{U}^\pi\beta$ is true at τ if "α until β" is true on a finite stretch of behavior which is in the linear time behavior of the program π.

The classical connectives $\supset$ and $\wedge$ are defined as usual. The derived modalities $\langle\pi\rangle$ and $[\pi]$ can be defined as follows: $\langle\pi\rangle\alpha \equiv \top\mathcal{U}^\pi\alpha$ and $[\pi]\alpha \equiv \neg\langle\pi\rangle\neg\alpha$. Furthermore, if we let $\Sigma = \{a_1, \ldots, a_n\}$, the $\mathcal{U}$ (until), $\bigcirc$ (next), $\Diamond$ and $\Box$ operators of LTL can be defined as follows: $\bigcirc\alpha \equiv \bigvee_{a\in\Sigma}\langle a\rangle\alpha$, $\alpha\mathcal{U}\beta \equiv \alpha\mathcal{U}^{\Sigma^*}\beta$, $\Diamond\alpha \equiv \top\mathcal{U}\alpha$, $\Box\alpha \equiv \neg\Diamond\neg\alpha$, where, in $\mathcal{U}^{\Sigma^*}$, Σ is taken to be a shorthand for the program $a_1 + \ldots + a_n$. Hence, LTL(Σ) is a

fragment of DLTL(Σ). As shown in (Henriksen and Thiagarajan 1999), DLTL(Σ) is strictly more expressive than LTL(Σ). In fact, DLTL has the full expressive power of the monadic second order theory of ω-sequences.

3 Action theories in Temporal ASP

Let $\mathcal{P}$ be a set of atomic propositions, the *fluent names*. A *simple fluent literal* l is a fluent name f or its negation $\neg f$. Given a fluent literal l, such that $l = f$ or $l = \neg f$, we define $|l| = f$. We denote by Lit_S the set of all simple fluent literals and, for each $l \in Lit_S$, we denote by $\bar{l}$ the complementary literal (namely, $\overline{p} = \neg p$ and $\overline{\neg p} = p$). Lit_T is the set of *temporal fluent literals*: if $l \in Lit_S$, then $[a]l, \bigcirc l \in Lit_T$ (for $a \in \Sigma$). Let $Lit = Lit_S \cup Lit_T \cup \{\bot\}$, where $\bot$ represents inconsistency. Given a (temporal) fluent literal l, $not\ l$ represents the default negation of l. A (temporal) fluent literal possibly preceded by a default negation, will be called an *extended fluent literal*.

In the following, to define our action language, we make use of a notion of *state*: a set of fluent literals. A state is said to be *consistent* if it is not the case that both f and $\neg f$ belong to the state, or that $\bot$ belongs to the state. A state is said to be *complete* if, for each fluent name $p \in \mathcal{P}$, either p or $\neg p$ belong to the state. The execution of an action in a state may possibly change the values of fluents in the state through its direct and indirect effects, thus giving rise to a new state.

Given a set of actions Σ, a *domain description* D *over* Σ is defined as a tuple $(\Pi, \mathcal{C})$, where Π is a set of laws (*action laws*, *causal laws*, *precondition laws*, etc.) describing the preconditions and effects of actions, and $\mathcal{C}$ is a set of *DLTL constraints*. While Π contains the laws that are usually included in a domain description, which define the executability conditions for actions, their direct and indirect effects as well as conditions on the initial state, $\mathcal{C}$ contains general DLTL constraints which must be satisfied by the intended interpretations of the domain description. As we will see, while the laws in Π allow the definition of local conditions, that must be satisfied by single states or by pairs of consecutive states, the DLTL constraints define more general conditions on the possible executions of actions, involving many different states. Let us first describe the laws occurring in Π.

The **action laws** in Π describe the immediate effects of actions. They are rules of the form:

$$\Box([a]l_0 \leftarrow t_1, \ldots, t_m, not\ t_{m+1}, \ldots, not\ t_n) \tag{1}$$

where l_0 is a simple fluent literal and the t_i's are either simple fluent literals or temporal fluent literals of the form $[a]l$. Its meaning is that executing action a in a state in which the conditions $t_1, \ldots, t_m$ hold and conditions $t_{m+1}, \ldots, t_n$ do not hold causes the effect l_0 to hold. Observe that, a temporal literal $[a]l$ is true in a state when the execution of action a in that state causes l to become true in the next state. For instance, we can describe the deterministic effect of the action *shoot* and *load* through the following action laws:

$$\Box([shoot]\neg alive \leftarrow loaded)$$

(the action of shooting the turkey makes the turkey dead if the gun is loaded) and

$$\Box[load]loaded$$

(loading the gun makes the gun loaded).

Non deterministic actions can be defined by making use of negation as failure in the body of action laws. As an example of non-deterministic action, consider the action of spinning the gun, after which the gun may be loaded or non-loaded:

$$\Box([spin]loaded \leftarrow \ not\ [spin]\neg loaded)$$
$$\Box([spin]\neg loaded \leftarrow \ not\ [spin]loaded)$$

Observe that, in this case, temporal fluent literals occur in the body of action laws.

Causal laws are intended to express "causal" dependencies among fluents. In Π we allow two kinds of causal laws: static causal laws and dynamic causal laws.

Static causal laws have the form:

$$\Box(l_0 \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n) \tag{2}$$

where the l_i's are simple fluent literals. Their meaning is: if $l_1, \ldots, l_m$ hold in a state and $l_{m+1}, \ldots, l_n$ do not hold in that state, than l_0 is caused to hold in that state.

Dynamic causal laws have the form:

$$\Box(\bigcirc l_0 \leftarrow t_1, \ldots, t_m, not\ t_{m+1}, \ldots, not\ t_n) \tag{3}$$

where the l_i's are simple fluent literals and t_i's are either simple or temporal fluent literals of the form $\bigcirc l_i$. Their meaning is: if $t_1, \ldots, t_m$ hold in a state and $t_{m+1}, \ldots, t_n$ do not hold in that state, than l_0 is caused to hold in the next state. Observe that, a precondition $t_i = \bigcirc l_i$ holds in a state when l_i holds in the next state.

For instance, the static causal law $\Box(frightened \leftarrow in\_sight, alive)$ states that the turkey being in sight of the hunter causes it to be frightened, if it is alive; alternatively, the dynamic causal law $\Box(\bigcirc frightened \leftarrow \bigcirc in\_sight, \neg in\_sight, alive)$ states that if the turkey is alive, it *becomes* frightened (if it is not already) when it *starts* seeing the hunter; but it can possibly become non-frightened later, due to other events, while still being in sight of the hunter[1].

Besides action laws and causal laws, which apply to all the states, we also allow for laws in Π which only apply to the initial state. They are called **initial state laws** and have the form:

$$l_0 \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n \tag{4}$$

where the l_i's are simple fluent literals. Observe that initial state laws unlike static causal laws, only apply to the initial state as they are not prefixed by the $\Box$ modality. As a special case, the initial state can be defined as a set of simple fluent literals. For instance, the initial state laws

$$alive. \quad \neg in\_sight. \quad \neg frightened.$$

define the initial state: $\{alive, \neg in\_sight, \neg frightened\}$.

Given the action laws, causal laws and initial state laws introduced above, all the usual ingredients of action theories can be introduced in Π. In particular, let us consider that case when $\bot$ can occur as a literal in the head of those laws.

[1] Shorthands like those in (Denecker et al. 1998) could be used, even though we do not introduce them in this paper, to express that a fluent or a complex formula is initiated (i.e. it is false in the current state and caused true in the next one).

Precondition laws are special kinds of action laws (1) with $\bot$ as effect. They have the form:

$$\Box([a] \bot \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n)$$

where $a \in \Sigma$ and the l_i's are simple fluent literals. The meaning is that the execution of an action a is not possible in a state in which $l_1, \ldots, l_m$ hold and $l_{m+1}, \ldots, l_n$ do not hold (that is, no state may results from the execution of a in a state in which $l_1, \ldots, l_m$ hold and $l_{m+1}, \ldots, l_n$ do not hold). An action for which no precondition law is given is regarded as being executable in any state.

State constraints which apply to the initial state or to all states can be obtained, respectively, when $\bot$ occurs in the head of initial state laws (4):

$$\bot \leftarrow t_1, \ldots, t_m, not\ t_{m+1}, \ldots, not\ t_n$$

or in the head of static causal laws (2)

$$\Box(\bot \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n)$$

The first one says that it is not the case that, in the initial state, $l_1, \ldots, l_m$ hold and $l_{m+1}, \ldots, l_n$ do not hold. The second one says that there is no state in which $l_1, \ldots, l_m$ hold and $l_{m+1}, \ldots, l_n$ do not hold.

As in (Lifschitz 1990; Kartha and Lifschitz 1994) we call *frame* fluents those fluents to which the law of inertia applies. The persistency of frame fluents from a state to the next one can be enforced by introducing in Π a set of laws, called **persistency laws**,

$$\Box(\bigcirc f \leftarrow f,\ not \bigcirc \neg f)$$
$$\Box(\bigcirc \neg f \leftarrow \neg f,\ not \bigcirc f)$$

for each simple fluent f to which inertia applies. Its meaning is that, if f holds in a state, then f still holds in the next state, unless its complement $\neg f$ is caused to hold. And similarly for $\neg f$. Note that, persistency laws are instances of the dynamic causal laws (3). In the following, for sake of conciseness, to include the above persistency laws for fluent f in a domain description, we will simply write **inertial f**.

For instance, we can capture the fact that $loaded$ is an inertial fluent (**inertial loaded**), by introducing the two persistency laws:

$$\Box(\bigcirc loaded \leftarrow loaded,\ not \bigcirc \neg loaded)$$
$$\Box(\bigcirc \neg loaded \leftarrow \neg loaded,\ not \bigcirc loaded)$$

The persistency of a fluent from a state to the next one can be blocked by the execution of an action which causes the value of the fluent to change. For instance, the execution of $load$ in a state where the gun is unloaded ($\neg loaded$) blocks the persistency of $\neg loaded$ to the next state as the action $load$ causes the gun to be $loaded$ as its immediate effect. Also, the persistency of the fluent $loaded$ is blocked by the execution of the action $spin$, which may nondeterministically cause the gun to become $loaded$ or $\neg loaded$, independently of the initial value of the fluent itself.

Although most fluents are inertial, and they do not change their values unless an action which affects their value is executed, there are also fluents which are not subject to the law of inertia. For instance, there are fluents which normally take a default truth value, as a

spring door which is normally closed

$$\Box(closed \leftarrow not \neg closed)$$

or other non inertial fluents, like a pendulum (see (Giunchiglia et al. 2004)), which moves from the leftmost position to the rightmost position and back and whose "default" behavior can be described by the action laws:

$$\Box(\bigcirc right \leftarrow \neg right,\ not \bigcirc \neg right)$$
$$\Box(\bigcirc \neg right \leftarrow right,\ not \bigcirc right)$$

Such default action laws play a role similar to that of default expressions in C^+ (Giunchiglia et al. 2004) and in $\mathcal{K}$ (Eiter et al. 2004). In such cases, persistency laws are not included in the domain description for non-inertial fluents.

As we have seen, the specification of the initial state by initial state laws may be, in general, incomplete. However, in this paper we want to reason about complete states so that the execution of an infinite sequence of actions gives rise to a linear model as defined in section 2. For this reason, we want to complete in all the possible ways the possibly incomplete initial states. We assume that, for each fluent f, Π contains the law:

$$f \leftarrow not\ \neg f$$
$$\neg f \leftarrow not\ f$$

whose effect is that either f is assumed to hold in the initial state, or $\neg f$ is assumed to hold. In the following, we assume that, for each fluent f, the set of laws introduced above for completing the initial state are implicitly included in Π, although, as we will see later, this assumption in general is not sufficient to guarantee that all the states are complete.

DLTL does not include *test actions* as specific kinds of actions. However, test actions are useful for checking the value of a fluent in a state, and they can be suitably defined. Given a simple fluent literal $l \in Lit_S$, we define a test action $l?$, for testing if l is true in the current state. Action $l?$ is executable in a state only if the literal l holds in that state (otherwise, the action is non executable):

$$\Box([l?]\bot \leftarrow not\ l)$$

The test actions can be regarded as atomic actions with no effects:

$$\Box([l?]f \leftarrow f) \qquad \Box([l?]\neg f \leftarrow \neg f)$$

for all fluent names f: all simple fluent literals keep their values after the execution of the test action $l?$. As we will see below, the presence of test actions is essential for the definition of interesting complex actions.

The second component of a domain description is the set $\mathcal{C}$ of *DLTL constraints*, which allow very general temporal conditions to be imposed on the executions of the domain description (we will call them extensions). Their effect is that of restricting the space of the possible executions. For instance, suppose that we want to add the condition that the hunter does not load the gun until the turkey is in sight. We can add in $\mathcal{C}$ the temporal constraint:

$$\neg loaded\, \mathcal{U}\ in\_sight$$

stating that the gun must not be loaded until the turkey is in sight. The addition of such

a temporal constraint to the domain description filters out all the executions of a domain description in which the gun is loaded before the turkey is in sight.

A temporal constraint can also require a complex behavior to be performed. The program

$$\pi = (\neg in\_sight?; wait)^*; in\_sight?; load; shoot \tag{5}$$

describes the behavior of the hunter who waits for a turkey until it appears and, when it is in sight, loads the gun and shoots. Actions $in\_sight?$ and $\neg in\_sight?$ are test actions, as introduced before. If the constraint

$$\langle(\neg in\_sight?; wait)^*; in\_sight?; load; shoot\rangle\top$$

is included in $\mathcal{C}$ then all the runs of the domain description which do not start with an execution of the given program are filtered out. For instance, an extension in which in the initial state the turkey is not in sight and the hunter loads the gun and shoots is not allowed. In general, the inclusion of a constraint $\langle\pi\rangle\top$ in $\mathcal{C}$ requires that there is an execution of the program π starting from the initial state.

Example 1

We can put together some of the laws introduced above to define the domain description for a variant of the Yale shooting problem. As in the Russian turkey problem, besides the action of loading the gun, shooting to the turkey and waiting, the hunter can execute the action of spinning the gun, after which we do not know whether the gun is loaded or not. In addition, we have that: (i) if the hunter is in sight and the turkey is alive, the turkey becomes frightened; (ii) the hunter cannot load the gun until the turkey is out of sight; (iii) the turkey can become in sight or out of sight (nondeterministically) during waiting.

Let $\Sigma = \{load, shoot, spin, wait\}$ and $\mathcal{P} = \{alive, loaded, in\_sight, frightened\}$. We define a domain description (Π,$\mathcal{C}$), where Π contains the following laws:

Immediate effects:

$\Box([shoot]\neg alive \leftarrow loaded)$
$\Box[load]loaded$
$\Box([spin]loaded \leftarrow\ not\ [spin]\neg loaded)$
$\Box([spin]\neg loaded \leftarrow\ not\ [spin]loaded)$
$\Box([wait]in\_sight \leftarrow\ not\ [wait]\neg in\_sight)$
$\Box([wait]\neg in\_sight \leftarrow\ not\ [wait]in\_sight)$

Causal laws:

$\Box(frightened \leftarrow in\_sight, alive)$

Initial state laws:

$alive. \quad \neg in\_sight. \quad \neg frightened.$

Precondition laws:

$\Box([load]\ \bot\leftarrow loaded)$

All fluents in $\mathcal{P}$ are inertial: **inertial** alive, **inertial** loaded, **inertial** in_sight, **inertial** frightened; and $\mathcal{C} = \{\neg loaded\ \mathcal{U}\ in\_sight\}$.

Given this domain description we may want to ask if it is possible for the hunter to execute a behavior described by program π in (5) so that the turkey is still alive after

that execution. The intended answer to the query $\langle\pi\rangle alive$ would be yes, since there is a possible scenario in which this can happen.

While we will make the answer to the above query more precise in the next section, by introducing the notion of extension of a domain description, let us point out that the action theory we have introduced is well suited to deal with infinite executions.

Example 2
This example describes a mail delivery agent, which repeatedly checks if there is mail in the mailbox of a and in the mailbox of b and then it delivers the mail to a or to b, if any; otherwise, it waits. Then, the agent starts again the cycle. The actions in Σ are: $begin$, $sense\_mail(a)$ (the agent verifies if there is mail in the mailbox of a), $sense\_mail(b)$, $deliver(a)$ (the agent delivers the mail to a), $deliver(b)$, $wait$ (the agent waits). The fluent names are $mail(a)$ (there is mail in the mailbox of a) and $mail(b)$. The domain description contains the following laws for a:

Immediate effects:

$\Box[deliver(a)]\neg mail(a)$
$\Box([sense\_mail(a)]mail(a) \leftarrow \ not\ [sense\_mail(a)]\neg mail(a))$

Precondition laws:

$\Box([deliver(a)] \perp\leftarrow \neg mail(a))$
$\Box([wait] \perp\leftarrow mail(a))$

Their meaning is (in the order) that: after delivering the mail to a, there is no mail for a any more; the action $sense\_mail(a)$ of verifying if there is mail for a, may (non-monotonically) cause $mail(a)$ to become true; if there is no mail for a, $deliver(a)$ is not executable; if there is mail for a, wait is not executable. The same laws are also introduced for the actions involving b.
All fluents in $\mathcal{P}$ are inertial: **inertial** mail(a), **inertial** mail(b). Observe that, the persistency laws for inertial fluents interact with the immediate effect laws above. The execution of $sense\_mail(a)$ in a state in which there is no mail for a ($\neg mail(a)$), may either lead to a state in which $mail(a)$ holds (by the second action law) or to a state in which $\neg mail(a)$ holds (by the persistency of $\neg mail(a)$).

$\mathcal{C}$ contains the following constraints:

$\langle begin\rangle\top$
$\Box[begin]\langle sense(a); sense(b); (deliver(a) + deliver(b) + wait); begin\rangle\top$

The first one means that the action $begin$ must be executed in the initial state. The second one means that, after any execution of action $begin$, the agent must execute $sense(a)$ and $sense(b)$ in the order, then either deliver the mail to a or to b or wait and, then, execute action $begin$ again, to start a new cycle.

We may want to check that if there is mail for a, the agent will eventually deliver it to a. This property, which can be formalized by the formula $\Box(mail(a) \supset \Diamond\neg mail(a))$, does not hold as there is a possible scenario in which there is mail for a, but the mail is repeatedly delivered to b and never to a. The mail delivery agent we have described is not fair.

As another example, consider the following one concerning a controlled system from the automotive domain.

Example 3
We describe an adaptation of the qualitative causal model of the "common rail" diesel injection system from (Panati and Theseider Dupré 2001) where:

- Pressurized fuel is stored in a container, the *rail*, in order to be injected at high pressure into the cylinders. We ignore in the model the output flow through the injectors. Fuel from the tank is input to the rail through a *pump*.
- A regulating system, including, in the physical system, a *pressure sensor*, a *pressure regulator* and an *Electronic Control Unit*, controls pressure in the rail; in particular, the pressure regulator, commanded by the ECU based on the measured pressure, outputs fuel back to the tank.
- The control system repeatedly executes the *sense_p* (sense pressure) action while the physical system evolves through internal events.

Examples of formulas from the model are contained in Π:

$\Box([pump\_weak\_fault]f\_in\_low$

shows the effect of the fault event $pump\_weak\_fault$. Flows influence the (derivative of) the pressure in the rail, and the derivative influences pressure, e.g.:

$\Box(p\_decr \leftarrow f\_out\_ok, f\_in\_low)$ $\quad$ $\Box(p\_incr \leftarrow f\_out\_very\_low, f\_in\_low)$
$\Box(p\_steady \leftarrow f\_out\_low, f\_in\_low)$ $\quad$ $\Box([p\_change]p\_low \leftarrow p\_ok, p\_decr)$
$\Box([p\_change]p\_ok \leftarrow p\_low, p\_incr)$ $\quad$ $\Box([p\_change]\bot \leftarrow p\_steady)$
$\Box([p\_change]\bot \leftarrow p\_decr, p\_low)$ $\quad$ $\Box([p\_change]\bot \leftarrow p\_incr, p\_ok)$

The model of the pressure regulating subsystem includes:

$\Box([sense\_p]p\_obs\_ok \leftarrow p\_ok)$ $\quad$ $\Box([sense\_p]p\_obs\_low \leftarrow p\_low)$
$\Box(f\_out\_ok \leftarrow normal\_mode, p\_obs\_ok)$ $\quad$ $\Box([switch\_mode]comp\_mode)$
$\Box(f\_out\_low \leftarrow comp\_mode, p\_obs\_ok)$
$\Box(f\_out\_very\_low \leftarrow comp\_mode, p\_obs\_low)$

with the obvious mutual exclusion constraints among fluents. Initially, everything is normal and pressure is steady: p_ok, p_steady, f_in_ok, f_out_ok, $normal\_mode$.

All fluents are inertial. We have the following temporal constraints in $\mathcal{C}$:

$\Box((p\_ok \wedge p\_decr) \vee (p\_low \wedge p\_incr) \supset \langle p\_change\rangle\top)$
$\Box(normal\_mode \wedge p\_obs\_low \supset \langle switch\_mode\rangle\top)$
$[sense\_p]\langle(\Sigma - \{sense\_p\})^*\rangle\langle sense\_p\rangle\top$
$\Box[pump\_weak\_fault]\neg\Diamond\langle pump\_weak\_fault\rangle\top$

The first models conditions which imply a pressure change. The 2nd one models the fact that a mode switch occurs when the system is operating in normal mode and the measured pressure is low. The 3rd one models the fact that the control system repeatedly executes $sense\_p$, but other actions may occur in between. The 4th one imposes that at most one fault may occur in a run.

Given this specification, we can, for instance, check that if pressure is low in one state, it will be normal in the 3rd next one, namely, that the temporal formula $\Box(p\_low \supset \bigcirc\bigcirc\bigcirc p\_ok)$ is satisfied in all the possible scenarios admitted by the domain description. That is, the system tolerates a weak fault of the pump — the only fault included in this model. In general, we could, e.g., be interested in proving properties that hold if at most one fault occurs, or at most one fault in a set of "weak" faults occurs.

As we have seen from the examples, our formalisms allows naturally to deal with infinite executions of actions. Such infinite executions define the models over which temporal formulas can be evaluated. Although in many cases (e.g. planning) we want to reason on finite action sequences, it is easy to see that any finite action sequence can always be represented as an infinite one. More precisely, this can be achieved by adding to the domain description an action *dummy*, and the constraints $\Diamond\langle dummy\rangle\top$ and $\Box[dummy]\langle dummy\rangle\top$ stating that action *dummy* is eventually executed and, from that point on, only action *dummy* is executed. In the following, we will restrict our consideration to infinite executions and we will assume that the dummy action is introduced when needed.

4 Temporal answer sets and extensions for domain descriptions

Given a domain description $D = (\Pi, \mathcal{C})$, the laws in Π are rules of a general logic program extended with a restricted use of temporal modalities. In order to define the extensions of a domain description, we introduce a notion of *temporal answer set*, extending the usual notion of *answer set* (Gelfond 2007). The extensions of a domain description will then be defined as the temporal answer sets of Π satisfying the integrity constraints $\mathcal{C}$.

In the following, for conciseness, we call "simple (temporal) literals" the "simple (temporal) fluent literals". We call *rules* the laws in Π, which have one of the two forms:

$$l_0 \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n \tag{6}$$

where the l_i''s are simple literals, and

$$\Box(l_0 \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n) \tag{7}$$

where the l_i's are simple or temporal literals, the first one capturing initial state laws, the second one all the other laws. To define the notion of extension, we also need to introduce rules of the form: $[a_1; \ldots; a_h](l_0 \leftarrow l_1, \ldots, l_m)$, where the l_i's are simple or temporal literals, which will be used to define the reduct of a program. The modality $[a_1; \ldots; a_h]$ says that the rule applies in the state obtained after the execution of actions $a_1, \ldots, a_h$. Conveniently, also the notion of temporal literal used so far needs to be extended to include literals of the form $[a_1; \ldots; a_h]l$, meaning that the simple fluent l holds after the execution of the sequence of actions $a_1, \ldots, a_h$.

As we have seen, temporal models of DLTL are linear models, consisting in an action sequence σ and a valuation function V, associating a propositional evaluation with each state in the sequence (denoted by a prefix of σ). We extend the notion of answer set to capture this linear structure of temporal models, by defining a partial temporal interpretation as a pair (σ, S), where $\sigma \in \Sigma^\omega$ and S is a set of literals of the form $[a_1; \ldots; a_k]l$, where $a_1 \ldots a_k$ is a prefix of σ.

Definition 1
Let $\sigma \in \Sigma^\omega$. A *partial temporal interpretation* (σ, S) *(over* σ*)* is a set of temporal literals of the form $[a_1; \ldots; a_k]l$, where $a_1 \ldots a_k$ is a prefix of σ, and it is not the case that both $[a_1; \ldots; a_k]l$ and $[a_1; \ldots; a_k]\neg l$ belong to S or that $[a_1; \ldots; a_k]\bot$ belongs to S (namely, S is a *consistent* set of temporal literals).

A temporal interpretation (σ, S) is said to be *total* if either $[a_1; \ldots; a_k]p \in S$ or $[a_1; \ldots; a_k]\neg p \in S$, for each $a_1 \ldots a_k$ prefix of σ and for each fluent name p.

Observe that a partial interpretation (σ, S) provides, for each prefix $a_1 \dots a_k$, a partial evaluation of fluents in the state corresponding to that prefix. The (partial) state $w^{(\sigma,S)}_{a_1 \dots a_k}$ obtained by the execution of the actions $a_1 \dots a_k$ in the sequence can be defined as follows:

$$w^{(\sigma,S)}_{a_1 \dots a_k} = \{l : [a_1; \dots; a_k] l \in S\}$$

We define the *satisfiability of a simple, temporal and extended literal t in a partial temporal interpretation (σ, S) in the state $a_1 \dots a_k$*, (written $S, a_1 \dots a_k \models t$) as follows:

$(\sigma, S), a_1 \dots a_k \models \top$
$(\sigma, S), a_1 \dots a_k \not\models \bot$
$(\sigma, S), a_1 \dots a_k \models l$ *iff* $[a_1; \dots; a_k] l \in S$, for a simple literal l
$(\sigma, S), a_1 \dots a_k \models [a] l$ *iff* $[a_1; \dots; a_k; a] l \in S$ or $a_1 \dots a_k, a$ is not a prefix of σ
$(\sigma, S), a_1 \dots a_k \models \bigcirc l$ *iff* $[a_1; \dots; a_k; b] l \in S$, where $a_1 \dots a_k b$ is a prefix of σ
$(\sigma, S), a_1 \dots a_k \models not\ l$ *iff* $(\sigma, S), a_1 \dots a_k \not\models l$

The satisfiability of rule bodies in a partial interpretation are defined as usual:
$(\sigma, S), a_1 \dots a_k \models t_1, \dots, t_n$ *iff* $(\sigma, S), a_1 \dots a_k \models t_1$ and ... and $(\sigma, S), a_1 \dots a_k \models t$
A rule $H \leftarrow Body$ is satisfied in a partial temporal interpretation (σ, S) if, $(\sigma, S), \varepsilon \models Body$ implies $(\sigma, S), \varepsilon \models H$, where ε is the empty action sequence.

A rule $\Box(H \leftarrow Body)$ is satisfied in a partial temporal interpretation (σ, S) if, for all action sequences $a_1 \dots a_k$ (including the empty one), $(\sigma, S), a_1 \dots a_k \models Body$ implies $(\sigma, S), a_1 \dots a_k \models H$.

A rule $[a_1; \dots; a_h](H \leftarrow Body)$ is satisfied in a partial temporal interpretation (σ, S) if $(\sigma, S), a_1 \dots a_h \models Body$ implies $(\sigma, S), a_1 \dots a_h \models H$.

We are now ready to define the notion of answer set for a set P of rules that does not contain default negation. Let P be a set of rules over an action alphabet Σ, not containing default negation, and let $\sigma \in \Sigma^\omega$.

Definition 2
A partial temporal interpretation (σ, S) is a *temporal answer set of P* if S is minimal (in the sense of set inclusion) among the partial interpretations over σ satisfying the rules in P.

In order to define answer sets of a program P possibly containing negation, given a partial temporal interpretation (σ, S) over $\sigma \in \Sigma^\omega$, we define the *reduct, $P^{(\sigma,S)}$, of P relative to* (σ, S) extending Gelfond and Lifschitz' transform (Gelfond and Lifschitz 1988) to compute a different reduct of P for each prefix $a_1, \dots, a_h$ of σ.

Definition 3
The *reduct, $P^{(\sigma,S)}_{a_1,\dots,a_h}$, of P relative to (σ, S) and to the prefix $a_1, \dots, a_h$ of σ* , is the set of all the rules

$$[a_1; \dots; a_h](H \leftarrow l_1, \dots, l_m)$$

such that $\Box(H \leftarrow l_1, \dots, l_m, not\ l_{m+1}, \dots, not\ l_n)$ is in P and $(\sigma, S), a_1, \dots, a_h \not\models l_i$, for all $i = m+1, \dots, n$. The *reduct $P^{(\sigma,S)}$ of P relative to* (σ, S) is the union of all reducts $P^{(\sigma,S)}_{a_1,\dots,a_h}$ for all prefixes $a_1, \dots, a_h$ of σ.

In essence, given a partial interpretation (σ, S) over σ, a different reduct is computed for each finite prefix of σ, that is, for each possible state corresponding to a prefix of σ.

Definition 4
A partial temporal interpretation (σ, S) is an *answer set of* P if (σ, S) is an answer set of the reduct $P^{(\sigma,S)}$.

The definition above is a natural generalization of the usual notion of answer set to programs with temporal rules. Observe that, σ has infinitely many prefixes, so that the reduct $P^{(\sigma,S)}$ is infinite as well as its answer sets. This is in accordance with the fact that temporal models are infinite.

In the following, we will devote our attention to those domain descriptions $D = (\Pi, \mathcal{C})$ such that Π has total temporal answer sets. We will call such domain descriptions *well defined domain descriptions*. As we will see below, total temporal answer sets can indeed be regarded as temporal models (according to the definition of model in Section 2). Although it is not possible to define general syntactic conditions which guarantee that the temporal answer sets of Π are total, this can be done in some specific case. It is possible to prove the following:

Proposition 1
Let $D = (\Pi, \mathcal{C})$ be a domain description over Σ, such that all fluents are inertial. Let $\sigma \in \Sigma^\omega$. Any answer set of Π over σ is a total answer set over σ.

This result is not surprising, since, as we have assumed in the previous section, the laws for completing the initial state are implicitly added to Π, so that the initial state is complete. Moreover, it can be shown that (under the conditions, stated in Proposition 1, that all fluents are inertial) the execution of an action in a complete state produces (non-deterministically, due to the presence of non-deterministic actions) a new complete state, which can be only determined by the action laws, causal laws and persistency laws executed in that state.

In the following, we define the notion of *extension* of a well defined domain description $D = (\Pi, \mathcal{C})$ over Σ in two steps: first, we find the temporal answer sets of Π; second, we filter out all the temporal answer sets which do not satisfy the temporal constraints in $\mathcal{C}$. For the second step, we need to define when a temporal formula α is satisfied in a total temporal interpretation (σ, S). Observe that a total answer set (σ, S) can be easily seen as a temporal model, as defined in Section 2. Given a total answer set (σ, S) we define the corresponding temporal model as $M_S = (\sigma, V_S)$, where $p \in V_S(a_1, \ldots, a_h)$ if and only if $[a_1; \ldots; a_h]p \in S$, for all atomic propositions p. We say that a total answer set S over σ satisfies a DLTL formula α if $M_S, \varepsilon \models \alpha$.

Definition 5
An *extension of a well-defined domain domain description* $D = (\Pi, \mathcal{C})$ *over* Σ is a (total) answer set (σ, S) of Π which satisfies the constraints in $\mathcal{C}$.

Notice that, in general, a domain description may have more than one extension even for the same action sequence σ: the different extensions of D with the same σ account for the different possible initial states (when the initial state is incompletely specified) as well as for the different possible effects of nondeterministic actions.

Example 4
Assume the dummy action is added to the Russian Turkey domain in Section 3. Given the infinite sequence $\sigma_1 = \neg in\_sight?; wait; in\_sight?; load; shoot; dummy; \ldots$, the

domain description has (among the others) an extension (σ_1, S_1) over σ_1 containing the following temporal literals (for sake of brevity, we write $[a_1; \ldots; a_n](l_1 \wedge \ldots \wedge l_k)$ to say that $[a_1; \ldots; a_n]l_i$ holds in S_1 for all i's):
$[\varepsilon](alive \wedge \neg in\_sight \wedge \neg frightened \wedge \neg loaded)$,
$[\neg in\_sight?](alive \wedge \neg in\_sight \wedge \neg frightened \wedge \neg loaded)$,
$[\neg in\_sight?; wait](alive \wedge in\_sight \wedge frightened \wedge \neg loaded)$,
$[\neg in\_sight?; wait; in\_sight?](alive \wedge in\_sight \wedge frightened \wedge \neg loaded)$,
$[\neg in\_sight?; wait; in\_sight?; load](alive \wedge in\_sight \wedge frightened \wedge loaded)$,
$[\neg in\_sight?; wait; in\_sight?; load; shoot](\neg alive \wedge in\_sight \wedge frightened \wedge loaded)$,
$[\neg in\_sight?; wait; in\_sight?; load; shoot; dummy](\neg alive \wedge in\_sight \wedge frightened \wedge loaded)$
and so on. This extension satisfies the constraints in the domain description and corresponds to a linear temporal model $M_{S_1} = (\sigma_1, V_S)$.

To conclude this section we would like to point out that, given a domain description $D = (\Pi, \mathcal{C})$ over Σ such that Π only admits total answer sets, a *transition system* (W, I, T) can be associated with Π, as follows:

- W is the set of all the possible consistent and complete states of the domain description;
- I is the set of all the states in W satisfying the initial state laws in Π;
- $T \subseteq W \times \Sigma \times W$ is the set of all triples (w, a, w') such that: $w, w' \in W$, $a \in \Sigma$ and for some total answer set (σ, S) of Π: $w = w^{(\sigma,S)}_{[a_1;\ldots;a_h]}$ and $w' = w^{(\sigma,S)}_{[a_1;\ldots;a_h;a]}$

Intuitively, T is the set of transitions between states. A transition labelled a from w to w' (represented by the triple (w, a, w')) is present in T if, there is a (total) answer set of Π, in which w is a state and the execution of action a in w leads to the state w'.

5 Reasoning tasks

Given a domain description $D = (\Pi, \mathcal{C})$ over Σ and a temporal *goal* α (a DLTL formula), we are interested in finding out the extensions of $D = (\Pi, \mathcal{C})$ satisfying/falsifying α. While in the next sections we will focus on the use of bounded model checking techniques for answering this questions, in this one, we show that many reasoning problems, including temporal projection, planning and diagnosis can be characterized in this way.

Let us come back to the shooting domain in Example 1. Suppose we want to know if there is a possible scenario in which the turkey is not alive after the action sequence $\neg in\_sight?, wait; in\_sight?, load, shoot$. This is an instance of the *temporal projection problem*, that we can solve by finding out an extension of the domain description which satisfies the temporal formula

$$\langle \neg in\_sight?; wait; in\_sight?; load; shoot \rangle \neg alive$$

The extension S_1 in Example 4 indeed satisfies the temporal formula above, since $\langle \neg in\_sight?; wait; in\_sight?; load;$ is true in the linear model $M_{S_1} = (\sigma_1, V_S)$ associated with the extension S_1.

As it is well known from the planning literature, planning problem can be formulated as a satisfiability problem (Giunchiglia and Traverso 1999). In case of complete state and

deterministic actions, the problem of finding a plan which makes the turkey not alive and the gun loaded, can be stated as the problem of finding out an extension of the domain description in which the formula $\Diamond(\neg alive \wedge loaded)$ is satisfied. Such an extension provides a plan for achieving the goal $\neg alive \wedge loaded$.

It must be observed, however, that, in presence of incomplete initial state and of nondeterministic actions, the problem of finding a conformant/universal plan which works for all the possible completions of the initial state and for all the possible outcomes of nondeterministic actions cannot be simply solved by checking the satisfiability of the formula above. The computed plan must also be tested to be a universal plan. Let us consider, for instance, the complex plan:

$$\pi = (\neg in\_sight?; wait)^*; in\_sight?; load; shoot$$

one can verify that such a plan is indeed a universal plan, by verifying that there is no extension of the domain description satisfying the formula

$$\langle(\neg in\_sight?; wait)^*; in\_sight?; load; shoot\rangle alive$$

In such a case, there is no execution of the plan after which the turkey is still alive. Whatever the initial values of unspecified fluents and whatever the effect of nondeterministic actions might be, the plan π achieves its goal.

As concerns diagnosis, let us consider the controlled system Example 3. Given the observation p_obs_low in a state, we can ask if there is an extension of the domain description which explains it. A *diagnosis* of the fault is a run from the initial state to a state in which p_obs_low holds and which does not contain previous fault observation in the previous states (Panati and Theseider Dupré 2000).

In general, we can compute a diagnosis of the fault obs_f by finding an extension of the domain description which satisfies the formula: $(\neg obs_1 \wedge \ldots \wedge \neg obs_n)\ \mathcal{U}\ obs_f$, where obs_1, $\ldots, obs_n$ are all the possible observations of fault. Here, p_obs_low is the only possible fault observation, hence a diagnosis for it is an extension of the domain description which satisfies $\Diamond p\_obs\_low$.

As concerns property verification, an example has been given in Example 2. We observe that the verification that a domain description D is well defined can be done by adding to the domain description a static law $\Box(undefined\_fluent \leftarrow not\ f \wedge not\ \neg f)$, for each fluent literal f, and by verifying that there are no estensions in which $\Diamond undefined\_fluent$ holds in the initial state.

Among the other reasoning task which can be addressed by checking the satisfiability/validity of formulas in a temporal action theory, we want to mention the verification problems arising from the area of multiagent protocol verification (Giordano et al. 2007), as well as the verification of the compliance of business processes to norms. We refer to (D'Aprile et al. 2010) for a formulation of this problem as a problem of reasoning about actions with temporal answer sets.

6 Model checking

The above verification and satisfiability problems can be solved by extending the standard approach for verification and model-checking of Linear Time Temporal Logic, based on

the use of Büchi automata. As described in (Henriksen and Thiagarajan 1999), the satisfiability problem for DLTL can be solved in deterministic exponential time, as for LTL, by constructing for each formula $\alpha \in DLTL(\Sigma)$ a Büchi automaton $\mathcal{B}_\alpha$ such that the language of ω-words accepted by $\mathcal{B}_\alpha$ is non-empty if and only if α is satisfiable. The size of the automaton can be exponential in the size of α, while emptiness can be detected in a time linear in the size of the automaton.

The validity of a formula α can be verified by constructing the Büchi automaton $\mathcal{B}_{\neg\alpha}$ for $\neg\alpha$: if the language accepted by $\mathcal{B}_{\neg\alpha}$ is empty, then α is valid, whereas any infinite word accepted by $\mathcal{B}_{\neg\alpha}$ provides a counterexample to the validity of α.

The construction given in (Henriksen and Thiagarajan 1999) is highly inefficient since it requires to build an automaton with an exponential number of states, most of which will not be reachable from the initial state. A more efficient approach for constructing a Büchi automaton from a DLTL formula makes use of a tableau-based algorithm (Giordano and Martelli 2006). The construction of the automaton can be done on-the-fly, while checking for the emptiness of the language accepted by the automaton. As for LTL, the number of states of the automaton is, in the worst case, exponential in the size if the input formula, but in practice it is much smaller.

LTL is widely used to prove properties of (possibly concurrent) programs by means of *model checking* techniques. The property is represented as an LTL formula φ, whereas the program generates a transition system (the model), which directly corresponds to a Büchi automaton where all the states are accepting, and which describes all possible computations of the program. The property can be proved as before by taking the product of the model and of the automaton derived from $\neg\varphi$, and by checking for emptiness of the accepted language.

In our case, given a *domain description* $(\Pi, \mathcal{C})$, we have shown how to define a transition system from Π. Thus, given a property φ formulated as a DLTL formula, we can check its validity by checking the unsatisfiability of $\mathcal{C} \cup \neg\varphi$ in the transition system.

In (Biere et al. 2003) it has been shown that, in some cases, model checking can be more efficient if, instead of building the product automaton and checking for an accepting run on it, we build only an accepting run of the automaton (if there is one). In our case, this means to look for an infinite path of the transition system satisfying $\mathcal{C} \cup \neg\varphi$. This technique is called *bounded model checking*, since it looks for paths whose length is bounded by some integer k, by iteratively increasing the length k until a model satisfying $\mathcal{C} \cup \neg\varphi$ is found (if one exists). More precisely, it considers infinite paths which can be represented as a finite path of length k with a back loop from state k to a previous state in the path. It can be easily shown that, if a Büchi automaton has an accepting run, it has an accepting run which can be represented in this way.

A bounded model checking problem can be efficiently reduced to a propositional satisfiability problem or to an ASP problem. Unfortunately, if no model exists, the iterative procedure will never stop. Thus it is a partial decision procedure for checking validity. Techniques for achieving completeness are described in (Biere et al. 2003).

In the next section, we address the problem of defining a translation of a domain description into standard ASP, so that bounded model checking techniques can be used to check if a temporal goal (a DLTL formula) is satisfiable in some extension of the domain description.

7 Translation to ASP

In this section, we show how to translate a domain description to standard ASP. In particular, we have run the translated domain descriptions in DLV (Leone et al. 2006).

A temporal model consists of an infinite sequence of actions and a valuation function giving the value of fluents in the states of the model. States are represented in ASP as integers, starting with the initial state 0. We will use the predicates `occurs(Action,State)` and `holds(Literal,State)`. Occurrence of exactly one action in each state must be encoded:

```
-occurs(A,S):- occurs(A1,S),action(A),action(A1),A!=A1,state(S).

occurs(A,S):- not -occurs(A,S),action(A),state(S).
```

Given a *domain description* $(\Pi, \mathcal{C})$, the rules in Π can be translated as follows.
Action laws $\Box([a]l_0 \leftarrow t_1, \ldots, t_m, not\ t_{m+1}, \ldots, not\ t_n)$ are translated to

$$holds(l_0, S') \leftarrow state(S), S' = S + 1, occurs(a, S), h_1 \ldots h_m, not\ h_{m+1} \ldots not\ h_n$$

where $h_i = holds(l_i, S')$ if $t_i = [a]l_i$ or $h_i = holds(l_i, S)$ if $t_i = l_i$.
Dynamic causal laws $\Box(\bigcirc l_0 \leftarrow t_1, \ldots, t_m, not\ t_{m+1}, \ldots, not\ t_n)$ are translated to

$$holds(l_0, S') \leftarrow state(S), S' = S + 1, h_1 \ldots h_m, not\ h_{m+1} \ldots not\ h_n$$

where $h_i = holds(l_i, S')$ if $t_i = \bigcirc l_i$ or $h_i = holds(l_i, S)$ if $t_i = l_i$.

Static causal laws are translated in the same way, while static causal laws without the $\Box$ in front will be evaluated in state 0.
Precondition laws $\Box([a]\ \bot \leftarrow\ l_1, \ldots, l_m, not\ \ l_{m+1}, \ldots, not\ \ l_n)$ are translated to ASP constraints

$$\leftarrow state(S), occurs(a, S), h_1 \ldots h_m, not\ h_{m+1} \ldots not\ h_n$$

where $h_i = holds(l_i, S)$.

As described in the previous section, we are interested in infinite models represented as *k-loops*, i.e. a finite sequence of states from 0 to k with a back loop from state k to a previous state. Thus we assume a bound k to the number of states.

The above rules will compute a finite model from state 0 to state $k+1$. To detect the loop, we must find a state j, $0 \leq j \leq k$, equal to state $k + 1$ This can be achieved by defining a predicate `eq(State1,State2)` and a predicate `next((State1,State2)` such that `next(i,i+1)` for $0 \leq i \leq k - 1$, and `next(k,j)`.

```
eq(S1,S2) :- state(S1), state(S2), not diff(S1,S2).
diff(S1,S2) :- state(S1), state(S2), fluent(F), holds(F,S1),
holds(-F,S2).
diff(S1,S2) :- diff(S2,S1).
next(S,SN) :- state(S),laststate(LS),S<LS, SN=S+1.
-next(LS,S) :- laststate(LS),next(LS,SS),state(S),state(SS),S!=SS.
next(LS,S) :- laststate(LS),state(S), S<=LS, not -next(LS,S).
:- laststate(LS), next(LS,S), SuccLS = LS+1, not eq(SuccLS,S).
```

The second and third rule of predicate $next$ state that there must be exactly one state next to state k, while the last constraint states that the state next to state k must be equal to state $k + 1$.

Given a domain description $(\Pi, \mathcal{C})$, we denote by $tr(\Pi)$ the set of rules containing the translation of each law in Π, as defined above, as well as the definitions of eq,*diff* and $next$. As we have said, a total answer set R of $tr(\Pi)$ represents an infinite model as a *k-loop*. The corresponding temporal model, $M_R = (\sigma_R, V_R)$, can be defined as follows:

$\sigma_R = a_1 a_2 \ldots a_j a_{j+1} \ldots a_{k+1} a_{j+1} \ldots a_{k+1} \ldots$

where $occurs(a_1, 0), occurs(a_2, 1), ..., occurs(a_{j+1}, j), ..., occurs(a_{k+1}, k),\ next(k, j)$ belong to R, and, for all proposition $p \in \mathcal{P}$:

$p \in V_R(a_1 \ldots a_h)$ if and only if $holds(p, h) \in R$, for $0 \leq h \leq k$

$p \in V_R(a_1 \ldots a_{k+1})$ if and only if $holds(p, j) \in R$.

We can show that there is a one to one correspondence between the temporal answer sets of Π and the answer sets of the translation $tr(\Pi)$. Let $(\Pi, \mathcal{C})$ be a well-defined domain description over Σ.

Theorem 1

- Given a temporal answer set (σ, S) of Π such that σ can be finitely represented as a finite path with a back loop, there is a total answer set R of $tr(\Pi)$ such that R and S correspond to the same temporal model.
- Given an answer set R of $tr(\Pi)$, there is a total temporal answer set (σ, S) of Π (that can be finitely represented as a finite path with a back loop) such that R and S correspond to the same temporal model.

The proof is omitted for lack of space.

Let us now come to the problem of evaluating a DLTL formula over the models associated with the answer sets of $tr(\Pi)$. To deal with DLTL formulas, we use the predicate `sat(alpha,S)`, to express satisfiability of a DLTL formula α in a state of a model. As in (Giordano and Martelli 2006) we assume that $until$ formulas are indexed with finite automata rather than regular expressions, by exploiting the equivalence between regular expressions and finite automata. Thus we have $\alpha \mathcal{U}^{\mathcal{A}(q)} \beta$ instead of $\alpha \mathcal{U}^{\pi} \beta$, where $\mathcal{L}(\mathcal{A}(q)) = [[\pi]]$. More precisely, let $\mathcal{A} = (Q, \delta, Q_F)$ be an ϵ-free nondeterministic finite automaton over the alphabet Σ without an initial state, where Q is a finite set of states, $\delta : Q \times \Sigma \rightarrow 2^Q$ is the transition function, and Q_F is the set of final states. Given a state $q \in Q$, we denote with $\mathcal{A}(q)$ an automaton $\mathcal{A}$ with initial state q. In the definition of predicate sat for *until* formulas, we refer to the following axioms (Henriksen and Thiagarajan 1999):

$$\alpha \mathcal{U}^{\mathcal{A}(q)} \beta \equiv (\beta \vee (\alpha \wedge \bigvee_{a \in \Sigma} \langle a \rangle \bigvee_{q' \in \delta(q,a)} \alpha \mathcal{U}^{\mathcal{A}(q')} \beta)) \quad (q \text{ is a final state of } \mathcal{A})$$

$$\alpha \mathcal{U}^{\mathcal{A}(q)} \beta \equiv (\alpha \wedge \bigvee_{a \in \Sigma} \langle a \rangle \bigvee_{q' \in \delta(q,a)} \alpha \mathcal{U}^{\mathcal{A}(q')} \beta) \quad (q \text{ is not a final state of } \mathcal{A})$$

In the translation to ASP, DLTL formulas will be represented with terms. In particular, the formula $\alpha \mathcal{U}^{\mathcal{A}(q)} \beta$ will be represented as `until(A,q,alpha,beta)`. Furthermore, we assume the automaton $\mathcal{A}$ to be described with the predicates `trans(A,Q1,Act,Q2)` defining the transitions, and `final(A,Q)` defining the final states. The definition of sat is the following:

fluent:
```
sat(F,S):- fluent(F), holds(F,S).
```
or:
```
sat(or(Alpha, Beta),S):- sat(Alpha,S).
sat(or(Alpha, Beta),S):- sat(Beta,S).
```
neg:
```
sat(neg(Alpha),S):- not sat(Alpha,S).
```

until:
```
sat(until(Aut,Q,Alpha,Beta),S):- final(Aut,Q),sat(Beta,S).
sat(until(Aut,Q,Alpha,Beta),S):-
    sat(Alpha,S),trans(Aut,Q,Act,Q1),occurs(Act,S),
    next(S,S1),sat(until( Aut,Q1,Alpha,Beta),S1).
```

Similar definitions can be given for derived connectives and modalities. For instance, the temporal formulas $\Diamond\alpha$, $\langle a\rangle\alpha$ and $[a]\alpha$ are represented, respectively, by the terms `eventually(t_alpha)`, `always(t_alpha)`, `diamond(a,t_alpha)` and `box(a,t_alpha)`, where `t_alpha` is the term encoding the formula α. The definition of `sat` for such formulas is the following:

eventually: `sat(eventually(Alpha),S):- sat(Alpha,S).`
eventually: `sat(eventually(Alpha),S):- next(S,SN),sat(eventually(Alpha),SN).`
$\langle a\rangle$: `sat(diamond(A,Alpha),S):- occurs(A,S),next(S,SN),sat(Alpha,SN).`
$[a]$: `sat(box(A,Alpha),S):- action(A), occurs(B,S),A!=B.`
$[a]$: `sat(box(A,Alpha),S):- occurs(A,S),next(S,SN),sat(Alpha,SN).`

Since states are complete, we can identify negation as failure with classical negation, thus having a two valued interpretation of DLTL formulas. We must also add a constraint `:- not sat(t_alpha,0)`, for each temporal constraint α in the domain description, where states are represented by numbers, 0 is the initial state and `t_alpha` is the term encoding the formula α. The presence of the constraint `:- not sat(t_alpha,0)`, in the translation of the domain description guarantees that α must be satisfied, as the negated formula `not sat(t_alpha,0)` is not allowed to be true in the answer set.

As an example, the encoding of the temporal constraint

$$\Box[begin]\langle sense(a); sense(b); (deliver(a) + deliver(b) + wait); begin\rangle\top$$

in Example 2, is given by the following rules:

```
:- not sat(neg(ev(neg(box(begin,until(aut,q1,true,true))))),0).
trans(aut,q1,sense(a),q2).
trans(aut,q2,sense(b),q3).
trans(aut,q3,deliver(a),q4).
trans(aut,q3,deliver(b),q4).
trans(aut,q3,wait,q4).
trans(aut,q4,begin,q5).
final(aut,q5).
```

The first rule encodes the constraint, while the following ones encode the definition of the automaton `aut`, which is equivalent to the regular expression indexing the until formula in the constraint.

It is easy to see that the computation of the satisfiability of a formula α in a given state depends only on a finite set of formulas consisting of the subformulas of α and the formulas derived from an *until* subformula. We say that a formula $\gamma\mathcal{U}^{\mathcal{A}(q')}\beta$ is *derived* from a formula $\gamma\mathcal{U}^{\mathcal{A}(q)}\beta$ if q' is reachable from q in $\mathcal{A}$.

It is possible to see that the definition of the predicate sat, as given above for the base cases (fluent, or, neg, until), provides a correct evaluation of the temporal formulas over the temporal models associated with the translation $tr(\Pi)$ of Π. Let $tr'(\Pi)$ be the set of

rules extending the rules in $tr(\Pi)$ with the definition of predicate `sat` above. Let $(\Pi, \mathcal{C})$ be a well-defined domain description over Σ. We can prove the following theorem.

Theorem 2
Let R be a total answer set of $tr(\Pi)$ and α a DLTL formula. The temporal model $M_R = (\sigma, V)$ associated with R satisfies α if and only if there is an answer set R' of $tr'(\Pi)$ such that R' extends R and `sat(t_alpha,0)` $\in R'$ (where `t_alpha` is the term representing the formula α).

Proof
Let $M_R = (\sigma, V)$. The theorem can be proved by showing that for all finite prefixes τ of σ, $M_R, \tau \models \alpha$ if and only if `sat(t_alpha,h)` $\in R'$, where h is the state of M_R obtained after the execution of the sequence of actions τ. The proof is by double induction on the length of the prefix τ and on the structure of α. $\square$

The above formulation of sat is indeed the direct translation of the semantics of DLTL, which is given for infinite models. Intuitively, we can show that it works also when the model is represented as a *k-loop*, by considering the case of *until* formulas. If S is a state belonging to the loop the goal $sat(\alpha \mathcal{U}^{\mathcal{A}(q)} \beta, S)$ can depend cyclically on itself. This happens if the only rule which can be applied to prove the satisfiability of $\alpha \mathcal{U}^{\mathcal{A}(q)} \beta$ (or one of its derived formulas in each state of the loop) is the second rule of *until*. In this case, $sat(\alpha \mathcal{U}^{\mathcal{A}(q)} \beta, S)$ will be undefined, which amounts to say that $\alpha \mathcal{U}^{\mathcal{A}(q)} \beta$ is not true. This is correct, since, if this happens, α must be true in each state of the loop, and β must be false in all states of the loop corresponding to final states of $\mathcal{A}$. Thus, by unfolding the cyclic sequence into an infinite sequence, $\alpha \mathcal{U}^{\mathcal{A}(q)} \beta$ will never be satisfied.

Given a domain description $D = (\Pi, \mathcal{C})$, the translation $tr(D)$ of D contains: the translation $tr(\Pi)$ of Π, the definition of the predicate sat and, for each temporal formula α in $\mathcal{C}$, the constraint `:- not sat(t_alpha,0).`

Let $(\Pi, \mathcal{C})$ be a well-defined domain description over Σ. Given Theorems 1 and 2 above, it can be proved that:

Corollary 1
There is a one to one correspondence between the extensions of the domain description D and the answer sets of its translation $tr(D)$ in ASP.

More precisely, each extension of D is in a one to one correspondence with an answer set of $tr(D)$, and both of them are associated with the same temporal model.

Given a temporal formula α, we may want to check if there is an extension of the domain description D satisfying it. To this purpose, as for the temporal formulas in $\mathcal{C}$, we add to the translation, $tr(D)$, of D the constraint `:- not sat(t_alpha,0)`, so that the answer sets falsifying α are excluded.

According to the bounded model checking technique, the search for an extension of the domain description satisfying α is done by iteratively increasing the length k of the sequence searched for, until a cyclic model is found (if one exists). On the other hand, validity of a formula α can be proved, as usual in model checking, by verifying that D extended with $\neg\alpha$ is not satisfiable. Let us consider, from Example 2, the property $\Box(mail(a) \supset \Diamond\neg mail(a))$ (if there is mail for a, the agent will eventually deliver it to a). This formula is valid if its negation $\Diamond\neg(mail(a) \supset \Diamond\neg mail(a))$ is satisfiable. We verify

the satisfiability of this formula, by adding to the translation of the domain description the constraint

```
:- not sat(ev(neg(impl(mail(b),ev(neg(mail(b))))))),0).
```

and looking for an extension. The resulting set of rules indeed has extensions, which can be found for $k \geq 3$ and provide counterexamples to the validity of the property above. For instance, the extension in which `next(0,1), next(1,2), next(2,3), next(3,0), occurs(begin,0), occurs(sense_mail(a),1), occurs(sense_mail(b),2), occurs(deliver_mail(a),3), mail(b)` holds in all states, and `mail(a)` only in states 2 and 3, can be obtained for $k = 3$.

8 Conclusions and related work

In this paper we have described an action language which is based on a temporal extension of ASP, in which temporal modalities are included within rules. In the action language general temporal DLTL formulas (possibly including regular programs indexing temporal modalities) are allowed in the domain description to constrain the space of possible extensions. The approach naturally deals with non-terminating computations and relies on bounded model checking techniques for the verification of temporal formulas.

In (Giordano et al. 2001) a temporal action theory based on the linear temporal logic DLTL has been developed and the temporal projection and planning problems are formalized as satisfiability problems in DLTL. In (Giordano et al. 2001) a monotonic solution to the frame problem was adopted, by introducing a completion construction. Default negation was not allowed in the body of action laws and causal laws. Due to the different treatment of the frame problem, even in the case when default negation is not present in the body of the laws in Π, the notion of extension defined here is not equivalent to the one in (Giordano et al. 2001). In particular, the formalization of causal rules in (Giordano et al. 2001) does not allow reasoning by cases. Also, the nonmonotonic solution proposed here has the advantage that it does not require action and causal laws to be stratified to avoid unexpected extensions which may arise when cyclic dependencies are present.

In the last decade, ASP has been shown to be well suited for reasoning about dynamic domains (Gelfond 2007). In (Baral and Gelfond 2000), Baral and Gelfond provide an encoding in ASP of the action specification language $\mathcal{AL}$, which extends the action description language $\mathcal{A}$ (Gelfond and Lifschitz 1993) by allowing static and dynamic causal laws, executability conditions and concurrent actions. The proposed approach has been used for planning (Phan Huy Tu et al. 2010) and diagnosis (Balduccini and Gelfond 2003). In (Eiter et al. 2000; Eiter et al. 2004) a logic-based planning language, $\mathcal{K}$, is presented which is well suited for reasoning about incomplete knowledge and is implemented on the top of the DLV system. In (Giunchiglia and Lifschitz 1998; Giunchiglia et al. 2004) the languages $\mathcal{C}$ and $\mathcal{C}^+$ provide an account of causality and deal with actions with indirect and non-deterministic effects and with concurrent actions. The action language defined in this paper can be regarded as a temporal extension of the language $\mathcal{A}$, which allows to deal with general temporal constraints, with complex actions and infinite computations. Similarly to $\mathcal{K}$, our action language allows for default negation within the body of the laws in Π. However, our action language does not deal with concurrent actions and incomplete knowledge.

Bounded model checking (Biere et al. 2003) is based on the idea to search for a counterexample of the property to be checked in executions which are bounded by some integer k. SAT based bounded model checking methods do not suffer from the state explosion problem as the methods based on BDDs. Helianko and Niemelä (Heljanko and Niemelä 2003) developed a compact encoding of bounded model checking of LTL formulas as the problem of finding stable models of logic programs. In this paper, we have extended the approach in (Heljanko and Niemelä 2003) for encoding bounded model checking of DLTL formulas in ASP. While the construction of a Büchi automaton (Henriksen and Thiagarajan 1999; Giordano and Martelli 2006) from a DLTL formula requires a specific machinery to deal with program expressions with respect to the usual construction for LTL, bounded LTL model checking can be naturally extended to deal with program expressions in temporal modalities, by directly encoding in ASP the recursive definition of the modalities.

The presence of temporal constraints in our action language is related to the work on temporally extended goals in (Dal Lago et al. 2002; Baral and Zhao 2007), which, however, is concerned with expressing preferences among goals and exceptions in goal specification.

$\mathcal{ESG}$ (Claßen and Lakemeyer 2008) is a second order extension of CTL* for reasoning about nonterminating Golog programs. In $\mathcal{ESG}$ programs include, besides regular expressions, nondeterministic choice of arguments and concurrent composition. The paper presents a method for verification of a first order CTL fragment of $\mathcal{ESG}$, using model checking and regression based reasoning. Because of first order quantification, this fragment is in general undecidable. DLTL (Henriksen and Thiagarajan 1999) is a decidable LTL fragment of $\mathcal{ESG}$ for which standard LTL model checking techniques can be adopted (Giordano and Martelli 2006). Satisfiability in DLTL is known to be PSPACE-complete, as for LTL (Henriksen and Thiagarajan 1999).

References

BACCHUS, F. AND KABANZA, F. 1998. Planning for temporally extended goals. *Annals of Mathematics and AI 22*, 5–27.

BALDUCCINI, M. AND GELFOND, M. 2003. Diagnostic reasoning with A-prolog. *Theory and Practice of Logic Programming 3,* 4-5, 425–461.

BARAL, C. AND GELFOND, M. 2000. Reasoning agents in dynamic domains. In *Logic-Based Artificial Intelligence*. 257–279.

BARAL, C. AND ZHAO, J. 2007. Non-monotonic temporal logics for goal specification. In *IJCAI 2007*. 236–242.

BIERE, A., CIMATTI, A., CLARKE, E. M., STRICHMAN, O., AND ZHU, Y. 2003. Bounded model checking. *Advances in Computers 58*, 118–149.

CLASSEN, J. AND LAKEMEYER, G. 2008. A logic for non-terminating Golog programs. In *Proc. KR 2008.*

DAL LAGO, U., PISTORE, M., AND TRAVERSO, P. 2002. Planning with a language for extended goals. In *Proc. AAAI02*.

D'APRILE, D., GIORDANO, L., GLIOZZI, V., MARTELLI, A., POZZATO, G., AND THESEIDER DUPRÉ, D. 2010. Verifying business process compliance by reasoning about actions. In *CLIMA 2010, LNCS 6245*.

DENECKER, M., THESEIDER DUPRÉ, D., AND BELLEGHEM, K. V. 1998. An inductive definitions approach to ramifications. *Electronic Trans. on Artificial Intelligence 2*, 25–97.

EITER, T., FABER, W., LEONE, N., PFEIFER, G., AND POLLERES, A. 2000. Planning under incomplete knowledge. In *Computational Logic 2000*. 807–821.

EITER, T., FABER, W., LEONE, N., PFEIFER, G., AND POLLERES, A. 2004. A logic programming approach to knowledge-state planning: Semantics and complexity. *ACM Trans. Comput. Log. 5,* 2, 206–263.

GELFOND, M. 2007. *Handbook of Knowledge Representation, chapter 7, Answer Sets*. Elsevier.

GELFOND, M. AND LIFSCHITZ, V. 1988. The stable model semantics for logic programming. In *Logic Programming, Proc. of the 5th Int. Conf. and Symposium*. 1070–1080.

GELFOND, M. AND LIFSCHITZ, V. 1993. Representing action and change by logic programs. *Journal of logic Programming 17*, 301–322.

GIORDANO, L. AND MARTELLI, A. 2006. Tableau-based automata construction for dynamic linear time temporal logic. *Annals of Mathematics and Artificial Intelligence 46,* 3, 289–315.

GIORDANO, L., MARTELLI, A., AND SCHWIND, C. 2001. Reasoning about actions in dynamic linear time temporal logic. *The Logic Journal of the IGPL 9,* 2, 289–303.

GIORDANO, L., MARTELLI, A., AND SCHWIND, C. 2007. Specifying and verifying interaction protocols in a temporal action logic. *Journal of Applied Logic (Special issue on Logic Based Agent Verification) 5*, 214–234.

GIUNCHIGLIA, E., LEE, J., LIFSCHITZ, V., MCCAIN, N., , AND TURNER, H. 2004. Nonmonotonic causal theories. *Artificial Intelligence 153,* 1-2, 49–104.

GIUNCHIGLIA, E. AND LIFSCHITZ, V. 1998. An action language based on causal explanation: Preliminary report. In *AAAI/IAAI*. 623–630.

GIUNCHIGLIA, F. AND TRAVERSO, P. 1999. Planning as model checking. In *Proc. The 5th European Conf. on Planning (ECP'99)*. 1–20.

HELJANKO, K. AND NIEMELÄ, I. 2003. Bounded LTL model checking with stable models. *TPLP 3,* 4-5, 519–550.

HENRIKSEN, J. AND THIAGARAJAN, P. 1999. Dynamic linear time temporal logic. *Annals of Pure and Applied logic 96,* 1-3, 187–207.

KABANZA, F., BARBEAU, M., AND ST-DENIS, R. 1997. Planning control rules for reactive agents. *Artificial Intelligence 95*, 67–113.

KARTHA, G. AND LIFSCHITZ, V. 1994. Actions with indirect effects (preliminary report). In *Proc. KR'94*. 341–350.

LEONE, N., PFEIFER, G., FABER, W., EITER, T., GOTTLOB, G., PERRI, S., AND SCARCELLO, F. 2006. The dlv system for knowledge representation and reasoning. *ACM Transactions on Computational Logic 7,* 3, 499–562.

LIFSCHITZ, V. 1990. Frames in the space of situations. *Artificial Intelligence 46*, 365–376.

PANATI, A. AND THESEIDER DUPRÉ, D. 2000. State-based vs simulation-based diagnosis of dynamic systems. In *Proc. ECAI 2000*.

PANATI, A. AND THESEIDER DUPRÉ, D. 2001. Causal simulation and diagnosis of dynamic systems. In *AI*IA 2001: Advances in Artificial Intelligence, LNCS 2175*.

PHAN HUY TU, TRAN CAO SON, GELFOND, M., AND MORALES, R. 2010. Approximation of action theories and its application to conformant planning. *Artificial Intelligence*.

PISTORE, M., TRAVERSO, P., AND BERTOLI, P. 2005. Automated composition of web services by planning in asynchronous domains. In *Proc. ICAPS 2005*. 2–11.